PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/00, 9/06

A1

(11) International Publication Number: WO 98/08160

(43) International Publication Date: 26 February 1998 (26.02.98)

(21) International Application Number: PCT/US97/12411

(22) International Filing Date: 16 July 1997 (16.07.97)

(30) Priority Data:
08/699,827 20 August 1996 (20.08.96) US

(71) Applicant: IDEA CORPORATION [US/US]; 19447 Pruneridge Avenue, Cupertino, CA 95014 (US).

(72) Inventor: CHANG, Po-Hua; 13043 Quito Road, Saratoga, CA 95070 (US).

(74) Agents: BEREZNAK, Bradley, J. et al.; Blakely, Sokoloff, Taylor & Zafman LLP, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

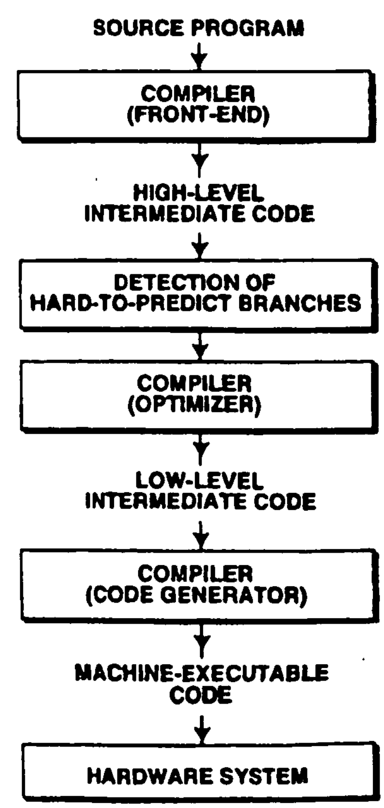
(81) Designated States: AL, AM, AT, AT (Utility model), AU (Petty patent), AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, CZ (Utility model), DE, DE (Utility model), DK, DK (Utility model), EE, EE (Utility model), ES, FI, FI (Utility model), GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (Utility model), SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: A METHOD FOR IDENTIFYING HARD-TO-PREDICT BRANCHES TO ENHANCE PROCESSOR PERFORMANCE

(57) Abstract

A method and apparatus for handling branch instructions contained within a source program includes applying a set of heuristics to classify each of the branch instructions in the source program as either a hard-to-predict type or a simple type of branch. A system implements a multi-heuristic branch predictor (21) comprising a large, relatively simple branch predictor (23) having many entries, to accommodate the majority of branch instructions encountered in a program, and a second, relatively small, sophisticated branch predictor (24) having a few entries. The sophisticated branch predictor (24) predicts the target addresses of the hard-to-predict branches. By mapping hard-to-predict branches to the sophisticated branch predictor (24), and easy-to-predict branches to the relatively simple branch predictor (23), overall performance is enhanced.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

A METHOD FOR IDENTIFYING HARD-TO-PREDICT BRANCHES TO ENHANCE PROCESSOR PERFORMANCE

FIELD OF THE INVENTION

The present invention relates generally to the field of branch prediction in microprocessors.

BACKGROUND OF THE INVENTION

Pipelining is a well-known technique whereby several instructions are overlapped in execution. Today, most microprocessors rely upon pipelining for improved, high-speed performance. A major effect of pipelining, however, is that it introduces data and control hazards, which can cause significant performance losses. For example, the ideal speedup from pipelining can be reduced by half due to pipeline stalls and other delays caused by branch penalties.

Branch instructions can be either unconditional, meaning that the branch is taken every time that the instruction is encountered in the program, or conditional, meaning that the branch is either taken or not taken, depending upon a condition. Most often, the instructions to be executed following a conditional branch are not known with certainty until the condition upon which the branch depends has been resolved. These types of branches can significantly reduce the performance of a pipeline processor since they may interrupt the steady supply of instructions to the execution hardware. Branch predictors attempt to predict the outcome of conditional branch instructions in a program before the branch instruction is executed. If a branch is mispredicted, all of the speculative work, beyond the point in the program where the branch is encountered, must be discarded. Therefore, a highly-accurate branch prediction mechanism is vital to a high-performance, pipelined processor.

The prior art is replete with different branch prediction schemes. A general overview of the problems associated with branch prediction, and the presentation of a number of solutions is provided in an article by J. Lee and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design", IEEE Computer (January 1984). An article authored by James E. Smith, entitled "A Study of Branch Prediction Strategies", IEEE (1981) discusses a variety of branch prediction techniques in terms of accuracy, costs and flexibility of use. A typical method of branch prediction utilizes a memory to store branch history information associated with the branch instruction. An example of this approach to branch prediction is found in U.S. Patent No. 5,142,634.

Many early implementations of branch predictors used simple history bits and counter-based schemes that provide branch prediction accuracy of about 85-90%. Attempts to improve upon the accuracy simple 2-bit counter schemes have included predictors that relate the sub-history information of a branch to the most recently executed branches via a shift register. An example of this approach is disclosed in the article entitled "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation", by Shien-Tai Pan, et al.

As the complexity of the branch prediction problem increases, so has the sophistication of branch predictors. By way of example, the article "Branch Classification: A New Mechanism for Branch Predictor Performance", by Po-Young Chang, et al., Proceedings from Micro-27 (December 1994) describes a hybrid predictor in which each component branch predictor predicts only those branches for which it is best suited. Other sophisticated approaches employ complicated branch prediction algorithms that try to predict whether or not a branch will be taken based upon a lot of history information. This category of branch predictions is exemplified by mechanisms disclosed in several papers by Tse-Yu Yeh and Yale N. Patt entitled,

"A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History" IEEE (1993); "Two-Level Adaptive Training Branch Prediction"; and "Alternative Implementations of Two-Level Adaptive Branch Prediction" ACM (1992).

One of the problems with sophisticated branch predictors is the large amount of the silicon space required for implementing the branch prediction hardware. This has presented microprocessor designers with a dilemma: either utilize a simple branch predictor (with limited accuracy) that occupies a small amount of area, or employ a sophisticated branch predictor (with higher accuracy) that takes up a relatively large amount of silicon space.

Thus, there exists an unsatisfied need for a way to optimize branch prediction.

SUMMARY OF THE INVENTION

The present invention improves the performance of computer systems which execute programs that include branch instructions. Because branch miss penalty is typically very large -- especially for deeply pipelined, out-of-order execution processors -- the invention advantageously identifies hard-to-predict branches and provides hardware optimization.

In one embodiment, the present invention comprises a method for handling branch instructions contained within a source program. The source program is processed by a compiler to produce a machine-executable code suitable for running on a hardware system. A compile-time semantic analysis of the source program identifies hard-to-predict branches. The analysis is performed by applying a set of heuristics to classify each of the branch instructions in the source program as either a hard-to-predict type or a simple type of branch. The heuristics cleanly differentiate between easy-to-predict (i.e., simple) and hard-to-predict branches so that the hardware can be optimized to achieve the goals of implementing branch prediction algorithms with minimum silicon space, while simultaneously achieving a high prediction accuracy.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features which are characteristic of the present invention are set forth in the appended claims. The invention itself, however, as well as other features and advantages, is best understood with reference to the detailed description which follows, read in conjunction with the accompanying drawings, and wherein:

Figure 1 is compilation process flow chart in accordance with one embodiment of the present invention.

Figure 2 illustrates the use of predication in accordance with one embodiment of the present invention.

Figure 3 illustrates the functional blocks of a front-end unit in accordance with an embodiment of the present invention.

Figure 4 shows a two-level adaptive prediction algorithm utilized in one embodiment of the present invention.

Figure 5 illustrates a branch prediction state machine.

DETAILED DESCRIPTION

The trend in microprocessor design is to employ increasingly sophisticated hardware branch predictors. But as the sophistication of the branch prediction hardware increases, so does the silicon space required to implement such prediction algorithms. Moreover, sophisticated branch predictors rely upon an abundance of history information to make accurate predictions. This means that a large enough history (i.e., until enough number of branches have been executed) must be accumulated in order to make an accurate prediction. For large programs, such as operating systems and database programs, sophisticated branch predictors may actually perform worse than simple 2-bit counter schemes, due to high branch predictor miss ratios and the high cost of initializing the branch history.

The present invention solves these problems by providing a system that implements a multi-heuristic branch prediction mechanism. The multi-heuristic branch predictor comprises a large, relatively simple branch predictor having many entries to accommodate the majority of branch instructions encountered in a program. There is also included a second, relatively small, sophisticated branch predictor having a few entries. The sophisticated branch predictor is utilized to predict the target addresses of a few branch instructions that are classified as hard-to-predict branches. The combined size of the large, simple branch predictor and the small, sophisticated branch predictor is much smaller than the large, sophisticated branch predictors commonly employed in today's computer systems.

By mapping hard-to-predict branches to the sophisticated branch predictor, and easy-to-predict branches to the relatively simple branch predictor, overall performance is better than if a large, sophisticated branch predictor is used to predict all of the branches in a program.

For large computer programs, the present invention advantageously reduces the number of branch prediction misses in the sophisticated branch predictor. This is accomplished through a compile-time semantic analysis of the source program that identifies hard-to-predict branches by applying a set of heuristics. In one embodiment of the present invention, this procedure is performed statically at compile time.

Figure 1 is a computation process flow chart for one embodiment of the present invention. As shown, the compiler front-end processes the source program and produces a high-level intermediate code therefrom. This high-level intermediate code is in a form that is more easily processed by the compiler optimizer at a subsequent stage. As will be appreciated by practitioners familiar with compilers and computer systems, the high-intermediate code contains full semantic information.

The next step in the process flow chart involves use of an algorithm to detect the hard-to-predict branches. This algorithm is run immediately after the compiler front-end phase has completed. The detection of the hard-to-predict branch instructions is performed by applying a set of heuristics at compile time to the high-level intermediate code.

In one embodiment, a branch is classified as a hard-to-predict type of branch if it is:

1. A conditional expression that involves an array element access (i.e., a[i]) or a pointer reference (i.e., *p); or

2. A loop that traverses a linked data structure (i.e., linked list); or

3. A loop that iterates a fixed number of iterations (e.g., >8).

It should be understood that the above-listed heuristics are the result of profiling important branches in SPEC95 CINT benchmark programs listed below in

Table I. The profiling compared actual branch direction with simulated prediction, with the performance of the branch predictor being determined on a per-branch basis.

Table I

BENCHMARK	INPUT
099.go	2stone19.in
124.m88ksim	dcrand
126.gcc	stmt.i
129.compress	bigtest.in
130.li	train.lsp
132.ijpeg	original_image.pm
134.perl	primes.pl
147.vortex	input.train/vortex.in

Branches that account for the largest number of branch miss predictions are listed in the Appendix. An analysis of common characteristics produced the above-listed set of heuristics. It is appreciated that other embodiments may apply a different set of heuristics. In still other implementations, a subset of the above-listed heuristics may be utilized.

Referring now to **Figure 3** there is shown a front-end cluster 17 of a microprocessor utilized in one embodiment of the present invention. Front-end cluster 17 is responsible for fetching and presenting instructions to the expand and dispersal network which includes the execution units of the processor. The main units of cluster 17 include an instruction fetch and rotate unit (IFR) 20, instruction

prefetch unit (IPF) 18, instruction paging cache unit (IPC) 19, and an integrated branch prediction (IBP) unit 21. IBP unit 21 comprises simple branch predictor 23 and sophisticated branch predictor 24 for handling easy and hard to predict branches, respectively. Memory hierarchy 30, which may include various levels of cache memory, is shown in **Figure** 3 coupled to units 18-20. The instruction prefetch unit 18 is responsible for prefetching instruction bundles from memory hierarchy 30 to feed the instruction pipeline. IFR unit 20 rotates and aligns instructions fetched from the hierarchy before presenting down to the expand and dispersal network of the microprocessor. IPC unit 19 performs the virtual-to-physical address translation of fetch addresses from IFR 20 and of prefetch addresses from IPF 18. Integrated branch prediction unit 21 dynamically predicts branches early in the instruction pipeline. BPT 21 makes a prediction on branches as they occur in the cached line fetched. If the prediction is "no taken" and more branches remain in the cache line fetched, then the BPT 21 remains on that cache line in the following cycles until no more branches remain or branches predicted "taken".

Instructions that have been classified as easy-to-predict are steered to simple branch predictor 23. Simple branch predictor 23 may comprise, for example, an ordinary 2-bit counter, or other well-known simple branch predictor circuits. An example of a saturating 2-bit up/down counter implementation of a prediction state machine is illustrated in **Figure 5**. A truth table for the state machine of Figure 5 is shown below in Table II.

Table II

OLD STATE	BRANCH PREDICTION	NEW STATE
00	0	00

01	0	00
10	0	01
11	0	10
00	1	01
01	1	10
10	1	11
11	1	11

It is appreciated that the state machine counter illustrated in **Figure 5** is incremented when a branch is taken until the counter saturates at state 11. The counter is decremented when a branch is not taken until the counter saturates at 00. Prediction is represented by the most significant bit of the state.

The sophisticated branch predictor unit 24 shown in **Figure 3** may comprise any of a variety of branch predictor circuits such as that shown in **Figure 4**. **Figure 4** illustrates a two-level adaptive branch predication algorithm that may consist of a set-associative cache, where each entry keeps a first-level history of a branch, and branch prediction logic. Each entry of predictor 24 keeps a 4-bit branch history. Each set of predictor 24 has a 16-entry pattern history table indexed by the branch history. Each pattern history table entry represents the branch prediction state based on a 2-bit, up/down, saturating counter, such as that shown in **Figure 5**. When predictions have started on an entry, a speculative copy of a 4-bit branch history is kept and used.

As shown in **Figure 4**, there are two processes that occur in parallel. First, the new history is used to access the pattern history table in order to generate a new prediction bit. The prediction is written into the pattern table for the next branch

prediction. Second, the old history is used to access the pattern history table in order to generate the state that needs to be updated. The updated state is then written back into the pattern history table.

In the example of Figure 4, the history of the entry to be updated is indicated as 0010, and the branch prediction is taken. The new history is 0101 and is used to index the pattern history table. The new prediction for the branch is obtained from the most significant bit, in this case 1, meaning predicted taken. The old history, 0010, is used to index the pattern history table to get the old state, 10. This old state, 10, is sent to the state machine along with the branch decision, and the new state, 11, is written back into the pattern history table. As previously explained, a 2-bit saturating, up/down counter may be used to update the states stored in the pattern history table.

Although the present invention contemplates utilizing a hint bit to classify a branch instruction as a hard-to-predict type of branch instruction, an alternative is to generate predicate execution code in place of some or all hard-to-predict branches. The predicated execution is the conditional execution of an instruction under the control of a predicate. By conditionally executing all instructions in a piece of code under guard of a condition (instead of requiring change of control flow), predication effectively eliminates branches from the code. This is beneficial on wide and deep pipelines where the affect of flushes due to branch miss predictions can cause bubbles in the execution pipeline. This gives rise to a large opportunity loss of instructions that could have been executed. In essence, predication converts a control dependence branch condition into a data dependence predicate bit.

For example, a processor architecture may define a 64-entry predicate register file consisting of one bit predicates. These predicates may be written by the results of compare operations, or other means. The execution of most operations may be

guarded by a controlling predicate. The guard can be specified on a third source specifier by each operation.

Referring back to **Figure 1**, for hard-to-predict type of branches, the compiler optimizer may convert some codes into predicated execution code. The information about the hard-to-predict branch is then passed to the low-level intermediate code so that the compiler code generator may use the information to mark each branch instruction whether it is hard to predict or easy to predict. For example, a hint bit in the instruction may be used to classify a branch instruction as the hard-to-predict type. Of course, the hardware system of support predicated execution and handling of the hard-to-predict branches in accordance with the present invention.

The hardware system can optimize the execution of hard-to-predict branches in several ways. In one case, the branch instruction has an associated hard-to-predict branch hint bit. In situations where the hint bit is set to indicate a hard-to-predict branch, a sophisticated branch predictor, such as a unit 24 in **Figure 3** is utilized for the prediction. The previously set forth heuristics are used by the compiler for the hard-to-predict branches, and for setting the branch hint bits for each branch instruction. Since the compiler has information on which branches are hard to predict, it can optionally convert some code to be executed in a predicated manner.

A flow diagram of the above-described process is shown in **Figure 2**. In **Figure 2**, decision block 11 identifies a branch instruction encountered in a program as being either easy or hard-to-predict type of branch. The easy-to-predict type of branch instruction is left unmarked as shown by block 12. Hard-to-predict branches are subjected to a further decision, i.e., whether or not to employ predication. This is shown in **Figure 2** by decision block 13. If predication is not to be used, the branch is simply marked (e.g., using a hint bit) as a hard-to-predict type of branch,

as shown by block 14. Conversion of the branch into predicated execution code is represented by block 15 in **Figure 2**.

CLAIMS

I claim:

1. A method of handling branches comprising the steps of:

(a) providing a source program having one or more branch instructions;

(b) processing the source program to produce a machine-executable code therefrom, with a set of heuristics being applied to classify each of the one or more branch instructions as a hard-to-predict type or a simple type of branch.

2. The method according to claim 1 further comprising the steps of:

(c) marking in the machine-executable code each of the one or more branch instructions with a bit that identifies the branch instruction as the hard-to-predict type of branch.

3. The method according of claim 2 further comprising the step of:

(d) running the machine-executable code on a hardware system that includes first and second branch predictors.

4. The method of claim 3 wherein step (d) comprises the steps of:

predicting a target address of the branch instruction of the hard-to-predict type of branch using the first branch predictor.

5. The method according to claim 3 wherein step (d) comprises the step of:

predicting the target address of the branch instruction of the simple type of branch using the second branch predictor.

6. The method according to claim 1 wherein the set of heuristics includes identification of a conditional expression that includes an array element access or a pointer reference.

7. The method according to claim 6 wherein the set of heuristics further includes a loop that traverses a linked data structure, or a loop that iterates more than a predetermined number of times.

8. The method according to claim 7 wherein the predetermined number of times is greater than 8.

9. The method according to claim 3 wherein the hardware system includes a microprocessor.

10. A method of handling branch instructions comprising the steps of:

(a) processing a source program that includes first and second branch instructions to produce an intermediate code;

(b) applying a set of heuristics to the intermediate code that classifies the first branch instruction as a hard-to-predict type, and the second branch instruction as a simple type;

(c) generating machine-executable code from the intermediate code, the machine-executable code including information which identifies the first branch instruction as the hard-to-predict type;

(d) running the machine-executable code on a hardware system that optimizes the execution of the first branch instruction.

11. The method according to claim 10 wherein the information that identifies the first branch instruction as the hard-to-predict type comprises a hint bit.

12. The method according to claim 11 wherein the hardware system includes a processor having a branch predictor that predicts a target address of the first branch instruction responsive to the hint bit.

13. The method according to claim 10 wherein step (c) comprises the step of:

converting the first branch instruction into predicated execution code.

14. A system for handling branch instructions of a computer program comprising:

a compiler that generates a machine-executable code from a source code listing of the computer program, the compiler including an algorithm that classifies a branch instruction as a hard-to-predict type of branch instruction according to one or more heuristics, the hard-to-predict type of branch instruction having an associated hint bit that is set by the compiler;

a processor that executes the machine-executable code generated by the compiler, the processor including first and second branch predictors that predict target addresses of the branch instructions, wherein the hint bit determines which of the first or second branch predictors is utilized for a particular branch instruction of the machine-executable code.

15. The system of claim 14 wherein the one or more heuristics includes identification of a conditional expression that contains an array element access or a pointer reference.

16. The system of claim 15 wherein the one or more heuristics further includes identification of a loop that traverses a linked data structure or a loop that iterates more than a predetermined number of times.

17. The system of claim 16 wherein the predetermined number of times is greater than 8.

18. A system for handling branch instructions of a computer program comprising:

a compiler having a front-end that processes the source code listing and produces a high-level intermediate code that contains semantic information therefrom, an algorithm that operates on the high-level intermediate code to classify a branch instruction as a hard-to-predict type of branch instruction according to one or more heuristics, an optimizer that converts the high-level intermediate code operated upon by the algorithm into a low-level intermediate code, a portion of the hard-to-predict type of branch instructions being converted into predicate execution code, a code generator that generates a machine-executable code from the optimized low-level intermediate code, a remaining portion of the hard-to-predict branch instructions being marked with a hint bit;

a processor that executes the machine-executable code generated by the compiler, the processor including first and second branch predictors that predict a target address of the branch instructions wherein the hint bit determines which of the

first or second branch predictors is utilized for a particular branch instruction of the machine-executable code.

19. The system of claim 18 wherein the one or more heuristics includes identification of conditional expression that contains an array element access or a pointer reference.

20. The system of claim 19 wherein the one or more heuristics further includes a loop that traverses a linked data structure or a loop that iterates more than a predetermined number of times.

21. The system of claim 20 wherein the predetermined number of times is greater than 8.

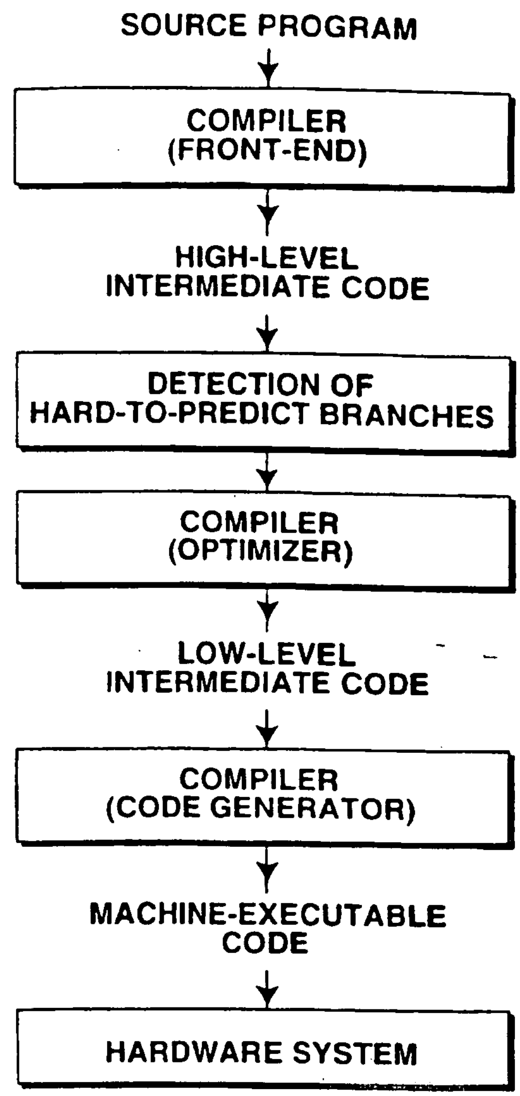

SOURCE PROGRAM
COMPILER (FRONT-END)
HIGH-LEVEL INTERMEDIATE CODE
DETECTION OF HARD-TO-PREDICT BRANCHES
COMPILER (OPTIMIZER)
LOW-LEVEL INTERMEDIATE CODE
COMPILER (CODE GENERATOR)
MACHINE-EXECUTABLE CODE
HARDWARE SYSTEM

Fig. 1

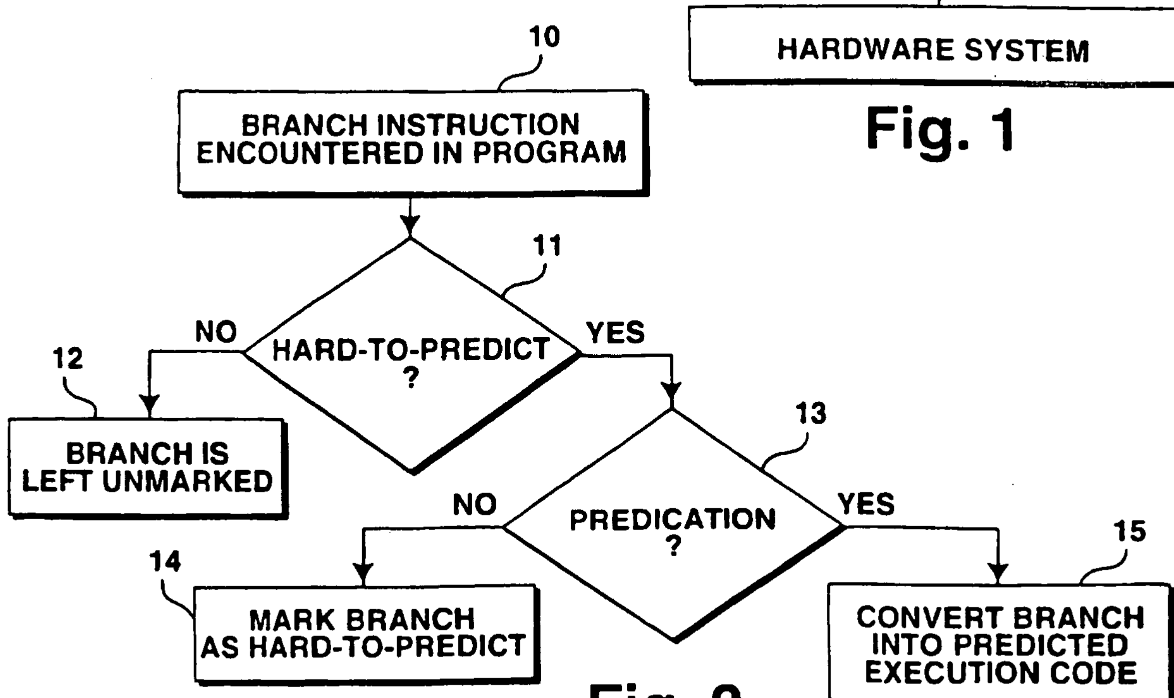
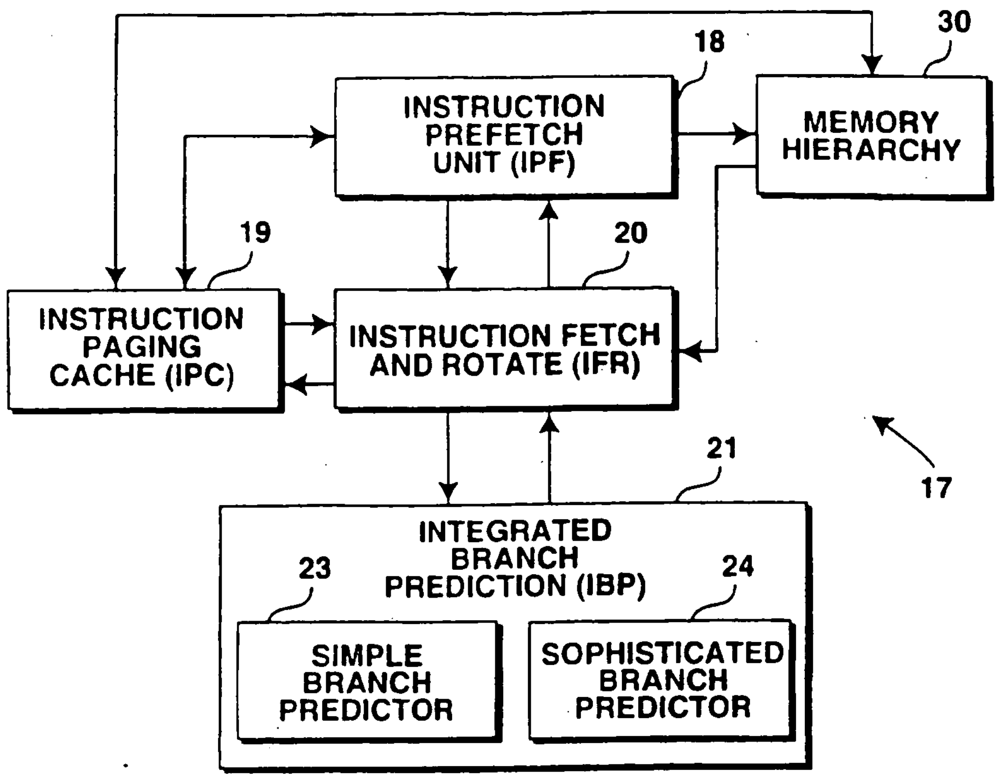
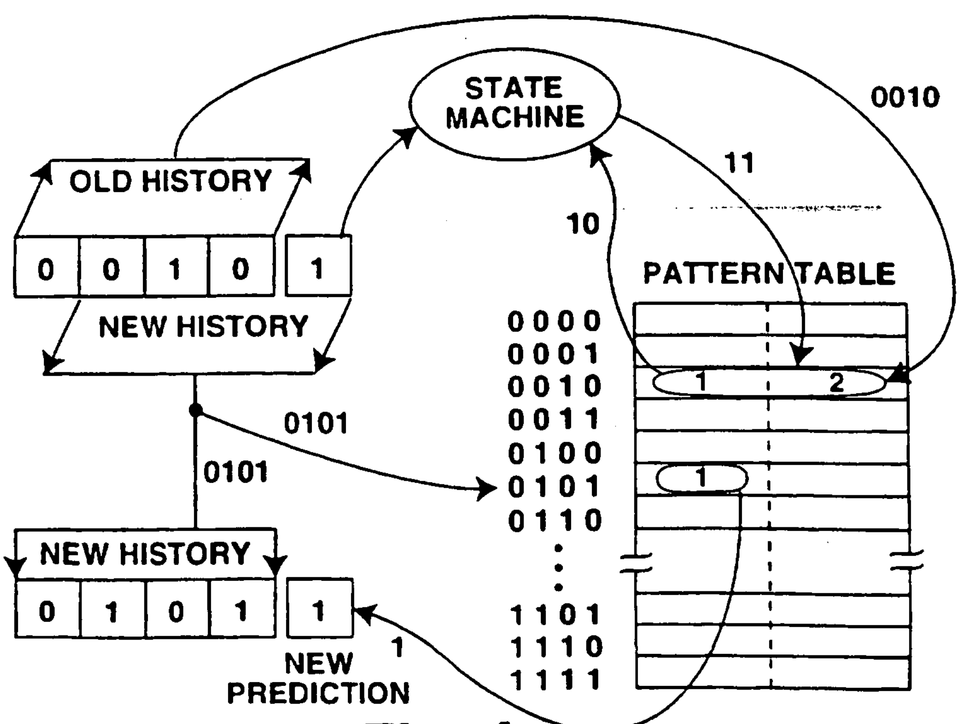

10
BRANCH INSTRUCTION ENCOUNTERED IN PROGRAM
11
HARD-TO-PREDICT ?
NO
YES
12
BRANCH IS LEFT UNMARKED
13
PREDICATION ?
NO
YES
14
MARK BRANCH AS HARD-TO-PREDICT
15
CONVERT BRANCH INTO PREDICTED EXECUTION CODE

Fig. 2



INSTRUCTION PREFETCH UNIT (IPF)
18
MEMORY HIERARCHY
30
INSTRUCTION PAGING CACHE (IPC)
19
INSTRUCTION FETCH AND ROTATE (IFR)
20
INTEGRATED BRANCH PREDICTION (IBP)
21
SIMPLE BRANCH PREDICTOR
23
SOPHISTICATED BRANCH PREDICTOR
24
17

Fig. 3



STATE MACHINE
0010
11
10
OLD HISTORY
0 0 1 0 1
NEW HISTORY
PATTERN TABLE
0000
0001
0010
0011
0100
0101
0110
1101
1110
1111
1
2
1
0101
0101
NEW HISTORY
0 1 0 1 1
1
NEW PREDICTION

Fig. 4

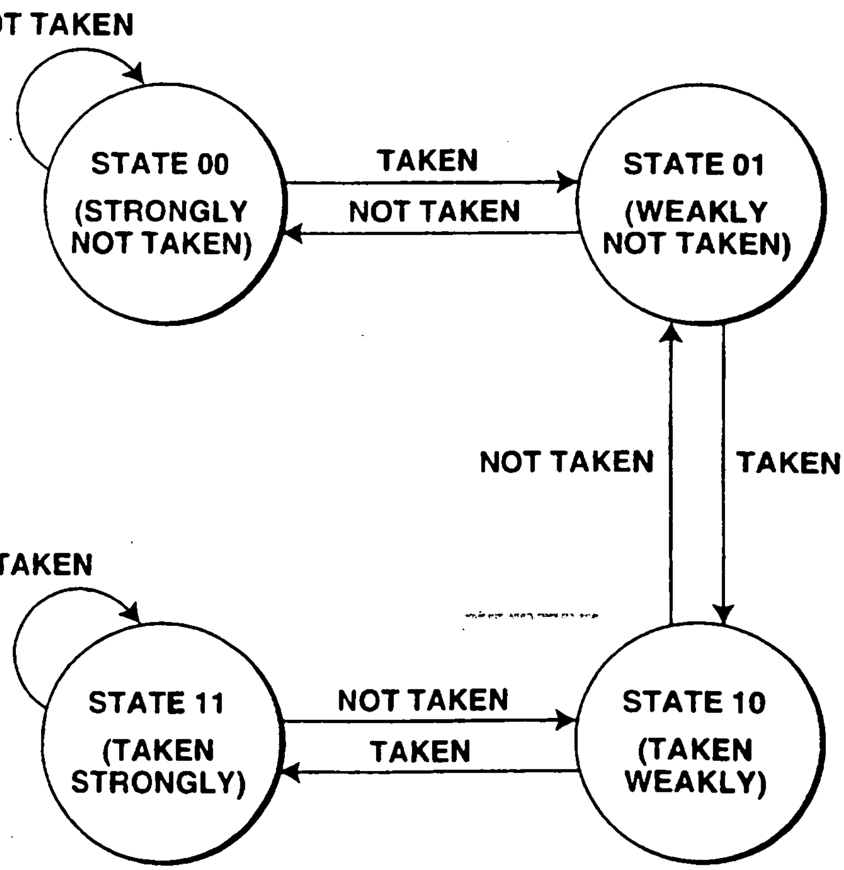

NOT TAKEN
STATE 00
(STRONGLY NOT TAKEN)
TAKEN
NOT TAKEN
STATE 01
(WEAKLY NOT TAKEN)
NOT TAKEN
TAKEN
TAKEN
STATE 11
(TAKEN STRONGLY)
NOT TAKEN
TAKEN
STATE 10
(TAKEN WEAKLY)

Fig. 5

INTERNATIONAL SEARCH REPORT	International application No. PCT/US97/12411

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 9/00, 9/06
US CL :395/580,581,582,583,584,585,586
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/580,581,582,583,584,585,586

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 4,287,559 A (EASLEY ET AL.) 01 September 1981, abstract, col. 8, lines 6-57, col. 34, lines 18-66.	1-13
Y	US 4,742,453 A (SHIBUYA) 03 May 1988, col. 8, lines 57 to col.9, lines 44.	1-13
Y	US 5,212,794 A (PETTIS ET AL.) 18 May 1993, col. 2, line 38 to col.4, line 57.	1-13
A	US 5,353,421 A (EMMA ET AL.) 04 October 1994, see the whole reference.	1-21
A,P	US 5,649,203 A (SITES) 15 July 1997, see the whole reference.	1-21

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
27 SEPTEMBER 1997	05 NOV 1997

Name and mailing address of the ISA/US	Authorized officer
Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231	ZARNI MAUNG Joni Hill
Facsimile No. (703) 305-3230	Telephone No. (703) 305-3900